УДК 004.42
ББК 32.972
Ф25

Фаррелл Б.
Ф25 Веб-компоненты в действии / пер. с англ. Д. А. Беликова. – М.: ДМК Пресс, 2020. – 462 с.: ил.

ISBN 978-5-97060-856-2

Один из основных факторов, способствующих трансформации интернета в последние годы, – широкое внедрение разработки пользовательского интерфейса на основе компонентов. В этой книге подробно описываются рабочие процессы, которые дают вам полный контроль над стилями и поведением компонентов и существенно упрощают их создание, совместное и повторное использование в проектах.

В первой части рассмотрено получение простого компонента с нуля. Вторая часть посвящена улучшению организации проекта. В третьей части освещаются принципы совместной работы с несколькими компонентами, позволяющей решать более сложные задачи. Для всех примеров предоставляется исходный код.

Издание предназначено для веб-разработчиков, имеющих опыт работы с HTML, CSS и JavaScript.

УДК 004.42
ББК 32.972

ISBN 978-1-617-29577-5 (англ.)
ISBN 978-5-97060-856-2 (рус.)

Моей удивительной жене,
которая пишет гораздо более захватывающие книги,
по сравнению с книгами о веб-разработке,
посвященные драконам и катастрофам

Оглавление

Содержание

Предисловие

Интернет прошел долгий путь. То, что началось три десятилетия назад как относительно простое средство публикации, совместного использования, обнаружения и потребления контента, превратилось в мощную и гибкую платформу приложений, поддерживающую невероятное количество вариантов использования. Между тем сфера его присутствия расширилась, и теперь выход в интернет осуществляется не только с настольных компьютеров, но и с устройств всех типов.

В результате этого постепенного преобразования мы, веб-разработчики, преследуем постоянно меняющуюся цель. Сегодняшние веб-сайты на несколько порядков сложнее по сравнению с их ранними предшественниками, и ожидания, связанные с пользовательским интерфейсом, значительно выросли.

К счастью, наш инструментарий также не стоял на месте. Сама веб-платформа приобрела сотни новых возможностей, а последующие поколения библиотек, фреймворков и инструментов постоянно совершенствуют современный уровень развития технологий, помогая нам удовлетворять растущие требования.

Одним из основных факторов, способствующих трансформации интернета в последние годы, стало широкое внедрение разработки пользовательского интерфейса на основе компонентов. Разделение нашей работы на компоненты, каждый из которых отвечает за структуру, стиль и поведение части пользовательского интерфейса, помогло нам управлять сложностью и создавать более сложные сайты.

Компоненты можно повторно использовать в каком-либо проекте или совместно в разных проектах, что повышает нашу эффективность. Дизайн-системы можно выразить в виде наборов готовых к использованию компонентов, обеспечивающих согласованность, которые позволяют командам сосредоточиться на конкретных потребностях продукта.

Популярные фреймворки помогли осуществить революцию компонентов, и сегодня большинство компонентов специфичны для конкретного фреймворка или библиотеки. Но параллельно предпринимались многолетние усилия по созданию мощной, нативной модели компонентов для веб-платформы.

Веб-компоненты – это общий термин для нового семейства функций веб-платформ, предлагающих прямую поддержку разработки на основе компонентов. Пользовательские элементы позволяют расширять словарь HTML, определяя собственные теги, которые легко работают со встроенными в браузер тегами и могут использоваться в одних и тех же местах, независимо от фреймворка. Технология Shadow DOM позволяет вам применять инкапсуляцию в нативном стиле, гарантируя, что CSS-правила компонента не будут непреднамеренно нарушать – и не будут нарушаться – форматирование страницы.

Вам, наверное, интересно, какие преимущества дают веб-компоненты по сравнению с компонентными моделями. С одной стороны, веб-компоненты обещают повысить совместимость, упрощая обмен компонентами даже между комплектами технологий. Модель общих компонентов также снижает риск блокировки, позволяя вам выполнять больше работы по мере изменения набора инструментов с течением времени.

Книга, которую вы сейчас держите в руках, исключительно своевременна. Путь к стандартизации и поддержка веб-компонентов претерпевали взлеты и падения, но я рад сообщить, что цель уже видна: все, кроме одного из популярных браузеров, уже поддерживают веб-компоненты, а когда состоится официальный релиз следующей версии Microsoft Edge, головоломка будет завершена.

Пользовательские элементы, Shadow DOM и другие функции веб-компонентов по своей природе являются низкоуровневыми примитивами. Некоторые разработчики будут использовать эти функции только косвенно, поскольку поддержка веб-компонентов во фреймворках увеличилась с ростом поддержки браузеров. Многие из самых популярных фреймворков теперь облегчают разработку и совместное использование веб-компонентов, и стал появляться целый новый класс инструментов, ориентированных на веб-компоненты.

Но вы также можете использовать функции веб-компонентов напрямую, по отдельности или сочетая их. Читая эту книгу, вы подробно изучите каждую функцию и то, как они связаны друг с другом, что даст вам возможность сделать правильный выбор для себя и своей команды.

Бен Фаррелл использовал веб-компоненты с момента их возникновения в самых разных приложениях. В ходе своей работы он накопил огромное количество ценных знаний и обнаружил множество эффективных шаблонов, которыми он поделится с вами на этих страницах.

Бен приводит примеры, демонстрируя разные концепции с помощью убедительных проектов, которые освещают реалистичные варианты использования. Вы, конечно, многому научитесь, но также непременно найдете здесь идеи и код, которые можно применить непосредственно в своих собственных проектах.

Решив заняться веб-компонентами и взяв эту книгу, вы сделали хороший выбор. Наслаждайтесь этим путешествием!

Грей Нортон,
технический руководитель /
менеджер проекта Polymer, GOOGLE

От автора

Для меня знакомство с веб-компонентами началось в 2013 году. Я помню, что работал над забавным небольшим проектом с использованием Angular версии 1 и изучал некоторые аспекты управления CSS и классами, которые Angular в то время плохо обрабатывал. Я знал, что мог бы легко сделать то, что мне нужно, в простом HTML, CSS и JavaScript, но Angular затруднял это только потому, что то, что я делал, находилось за пределами проторенных троп.

Примерно в это же время я почувствовал, что действительно начинаю овладевать Angular, поэтому написал в блоге несколько постов о некоторых интересных, нетипичных подходах. Тогда же волнение по поводу Angular стало угасать, и только начиналось волнение по поводу React.

Честно говоря, я был разочарован. Я долго смотрел на цикл, в котором чувствовал себя пойманным в ловушку. В течение всего двух или трех лет я постоянно учился и получал хорошие знания по фреймворкам JS. Ни один из этих фреймворков не был совместим друг с другом. Я дошел до того, что почувствовал, что действительно могу сосредоточиться на своем проекте без фреймворка на заднем плане, но затем неожиданно появилось нечто новое, что заставило меня почувствовать, что мне нужно вернуться на круги своя.

В то же самое время Google была выпущена библиотека Polymer как очень ранняя и нестабильная версия. Создание отдельных компонентов, которые могли бы существовать где угодно, звучало как удивительное обещание. Первоначально мне нравилось то, чего она пыталась достичь, но API, предшествующий первой версии, который постоянно менялся, и тот факт, что я заменял свой рабочий процесс еще одним фреймворком, заставил меня все переосмыслить. Я начал изучать предлагаемые веб-стандарты, которые сделали возможным создание библиотеки Polymer, и увидел огромный потенциал. Я понял, что это была не библиотека Polymer, которой я восхищался, – в действительности это были веб-компоненты.

Я начал вести блог и дискуссии о веб-компонентах. Примерно в это же время присоединился к Adobe. Это было важно, потому что моя команда работала над небольшими прототипами с одним, может быть, двумя разработчиками проекта. Это означало, что я мог экспериментировать с технологией и инструментами по своему выбору. Почти для каждого проекта я продолжал продвигать веб-компоненты, экспериментируя и постоянно улучшая рабочий процесс для работы с ними.

Конечно, это было непросто. Иногда я полностью был лишен почвы под ногами! Поскольку веб-компоненты стали стандартом, которым они являются сегодня, мы увидели, что изменение API и функции стали устаревшими, но у меня не было выбора, потому что мне действительно нравится работать как можно ближе к браузеру, используя только

HTML, JS и CSS, и я рассматривал веб-компоненты как средство обеспечения структуры своих проектов, а не для того, чтобы они превратились в спагетти-код.

Я еще не был полностью убежден в жизнеспособности веб-компонентов. С одной стороны, я пока не использовал Shadow DOM. Я не хотел увлекаться чем-то, что поддерживала только Google, у которой была сомнительная поддержка полифилов. Но затем веб-компоненты появились в браузере Safari, и Mozilla также пообещал, что будет поддерживать их. Вишенкой на торте стал момент, когда браузеры начали поддерживать модули JS и импорт нативно, и я смог правильно разделить код и, что более важно, HTML и CSS. Когда все это произошло, я знал, что веб-компоненты начинают реализовывать свой потенциал.

Конечно, все происходило очень медленно в течение нескольких лет. Многие разработчики, которые изначально были в восторге от веб-компонентов, потеряли терпение, и я не виню их. Сначала я обратился к издательству Manning по поводу книги о веб-компонентах – до того, как произошли некоторые важные ключевые события, например когда крупные компании-разработчики популярных браузеров объединились, чтобы завершить версию спецификации номер 1. В то время Manning не было уверено, особенно из-за того, что книги в этой области не публиковались, поскольку было неизвестно, чем все это закончится.

Был ли я настроен слишком оптимистично или просто провел с ними достаточно времени, чтобы узнать потенциал веб-компонентов, но издательство связалось со мной через год, чтобы сделать еще одно предложение. Даже тогда, в начале 2018 года, дело все равно могло бы принять дурной оборот, если бы другие компании-разработчики браузеров решили пойти на попятную. Кроме того, в то время я не подходил к разработке веб-компонентов так, как это делало большинство разработчиков, используя импорт HTML в качестве отправной точки. Тем не менее на протяжении этой книги класс LitElement от команды Polymer начал действовать в очень похожей со мной манере, используя шаблонные литералы для хранения разметки и стиля. Это, в сочетании с поддержкой веб-компонентов, когда над ними работала и компания Microsoft, осенью 2018 года позволило мне вздохнуть с облегчением, зная, что подходы, описанные в моей книге, идут в ногу с настоящим и будущим веб-компонентов. Я определенно продолжу совершенствовать свой рабочий процесс, по мере того как новые функции появляются в браузере и придумываются сообществом, но я очень рад нынешнему положению веб-компонентов, поскольку моя книга скоро будет опубликована. И конечно же, мне не терпится поделиться всем этим с читателями!

Благодарности

Данная книга была бы невозможна без всех тех удивительных людей, которые помогали мне на протяжении этого пути. Я хочу поблагодарить своих друзей из Северной Каролины и замечательных людей, которые проводят и посещают конференцию NCDevCon, за то, что они почти постоянно слушали мои доклады о веб-компонентах в Yammer. В частности, я хотел бы поблагодарить Эдриана Помилио за то, что он поразил меня своим выступлением в 2011 году, в котором были показаны пользовательские элементы, прежде чем они стали чем-то особенным.

Я также хотел бы поблагодарить членов команды GE Design System за то, что они были моими «сообщниками» в этом деле, в то время когда веб-компоненты были абсолютным новшеством, и мы были уверены, что все остальные считают нас безумцами. В частности, я хотел бы поблагодарить Мартина Рэгга, Джеффа Райхенберга и Джона Роджерсона за то, что они копались со мной в деталях при написании этой книги о новом способе создания сайтов. Еще хотел бы поблагодарить команду Google Polymer за помощь и руководство в течение этого времени, а также их технического руководителя Грея Нортона за написание предисловия к книге.

В Adobe я хотел бы поблагодарить всю команду Adobe Design (и за ее пределами) за поддержку и искреннюю радость по поводу публикации моей первой книги.

Конечно, моя жена Ребекка Гомес Фаррелл не только поддерживала меня, но и сама оказалась замечательным писателем и редактором. Помимо того что она принесла мне крепкий напиток, когда он мне понадобился, она помогла новому писателю стать лучше, давая стоящие, профессиональные советы.

Я хотел бы поблагодарить редакционную команду издательства Manning, в которую входят редакторы-консультанты по аудитории Кристен Уоттерсон, Кевин Харрелд и Ребекка Райнхарт, а также редактор-консультант по технической аудитории Дуглас Дункан, технический корректор Мэтью Уэлк, редактор по производству Энтони Калькара, редактор Ребекка Деуэль-Гальегос и корректор текста Тиффани Тейлор. Наконец, я хотел бы поблагодарить рецензентов, чьи отзывы и понимание сыграли важную роль в формировании этой книги, в том числе Альберто Чарланти, Алисию Бейкер, Бирну Себарте, Клайва Харбера, Дэниела Купера, Эрнана Гарсия, Джеймса Карелла, Джона Ларсена, Хуана Асенсио, Джастина Каллеха, Оливера Ковача, Пьетро Маффи, Рональда Бормана, Рассела Доуна Кахолес, Райана Барроуз, Серхио Арбео, Стефана Троста, Томаса Оверби Хансена, Тимоти Р. Кейна и Кумара С. Унникришнана (TR Technology & Ops).

Об этой книге

Книга «*Веб-компоненты в действии*» не диктует, какие подходы должны использовать разработчики. Вместо того чтобы рассказывать читателям, что делать, я использую более исследовательский подход, чтобы охватить основы веб-компонентов. Вы должны признать, что хотя эксперты могут сказать вам, что такое хороший рабочий процесс на сегодняшний день, захватывающий момент касательно стандартов состоит в том, что их можно создавать таким образом, которого никто не ожидает.

В этой книге я стремлюсь дать вам отличные идеи и рабочие процессы для начала работы. Я также надеюсь дать вам знания для дальнейшего использования веб-компонентов, способами, которые я еще не рассматривал, и для проектов, с которыми я не сталкивался.

Кому следует прочитать эту книгу

Данная книга предназначена для веб-разработчиков, которые интересуются веб-компонентами и хотят узнать больше о стоящих за ними стандартах и о том, как они объединяются с другими веб-технологиями для создания автономных компонентов или приложений.

Она также подходит для разработчиков, которым нужны идеи о том, как освободиться от сложных фреймворков или библиотек и вернуться к написанию простого HTML, JS и CSS без каких-либо шагов сборки.

Как организована эта книга: дорожная карта

Эта книга состоит из трех частей, охватывающих 15 глав и приложение.

В первой части приводятся первые шаги, описывающие получение простого компонента с нуля:

- в первой главе описывается, что имеется в виду, когда речь идет о веб-компонентах и различных стандартах, которые объединяются для их создания;
- вторая глава рассказывает о создании самого первого веб-компонента, а также знакомит вас с концепциями минимума, необходимыми для создания чего-то полезного;
- третья глава выводит минимальный компонент на следующий уровень, делая его многоразовым;
- в четвертой главе подробно описывается API веб-компонентов и жизненный цикл в сравнении их с другими API и жизненными циклами, с которыми вы, возможно, сталкивались;
- в пятой главе вы познакомитесь с модулями для более подходящего повторного использования кода и организации проекта.

Вторая часть основана на минимальном компоненте и охватывает концепции улучшения рабочего процесса разработчика и организации проекта:

- шестая глава подробно описывает использование модулей для разделения и импорта логики представления, такой как HTML и CSS, для лучшей организации вашего компонента;
- седьмая глава посвящена альтернативному, но не предпочтительному способу организации вашего компонента с помощью HTML-импорта, разбивая его на части, которые также относятся к другим аспектам веб-компонентов;
- восьмая глава знакомит вас с технологией Shadow DOM и рассказывает о ее пользе для защиты и инкапсуляции вашего компонента;
- в девятой главе мы продолжаем изучение Shadow DOM, чтобы охватить его CSS-аспекты;
- в десятой главе исследуются проблемы, которые могут возникнуть у разработчиков веб-компонентов с CSS в Shadow DOM, и способы избежать их или преодолеть.

Третья и последняя часть посвящена совместной работе с несколькими компонентами, чтобы создать нечто большее:

- в одиннадцатой главе рассматриваются освещенные ранее концепции, которые используются для создания нового, более отточенного компонента, основанного на уже созданных дочерних компонентах;
- в двенадцатой главе мы продвигаем этот абсолютно новый компонент, чтобы быть более готовым к промышленной эксплуатации благодаря применению инструментов сборки, которые позволяют использовать его в старых браузерах, не поддерживающих веб-компоненты;
- в тринадцатой главе мы дополняем этот компонент, написав для него тесты, которые выполняются в трех разных контекстах, чтобы изучить различные варианты, доступные для разработчиков веб-компонентов;
- в четырнадцатой главе обсуждается передача сообщений между вашими компонентами и подробно рассматривается распространенный шаблон проектирования;
- в пятнадцатой главе рассказывается о будущем веб-компонентов, а также о возможностях, которые они могут предоставить сегодня, скрывая сложность и делая все, от видеоэффектов в реальном времени до смешанной реальности, более простым в использовании.

Наконец, в приложении рассказывается о новых функциях Java Script (ES6/ES2015) и о том, как они помогают веб-компонентам.

О коде

Исходный код предоставляется для всех примеров в этой книге и доступен для скачивания с веб-сайта издательства Manning по адресу www.manning.com/books/web-components-in-action и из репозитория GitHub,

который можно найти по адресу https://github.com/bengfarrell/webcomponentsinaction. Репозиторий организован в папки для каждой главы, и в них обычно находятся вложенные папки для каждого раздела. Исключения составляют случаи работы над большим примером, охватывающим всю главу.

Код можно выполнить только с помощью браузера, и его не нужно компилировать до тех пор, пока вы не дойдете до следующих глав, посвященных инструментам сборки. Как правило, для запуска соответствующего HTML-файла, который служит примером, понадобится простой HTTP-сервер, но только для того, чтобы решить проблемы, связанные с разными источниками.

В этой книге содержится множество примеров исходного кода, как в виде пронумерованных листингов, так и встроенных в обычный текст. В обоих случаях исходный код форматируется `шрифтом фиксированной ширины`, `подобным этому`, чтобы отделить его от обычного текста. Иногда код также выделяется **`жирным шрифтом`**, чтобы выделить код, который изменился по сравнению с предыдущими шагами в этой главе, например когда к существующей строке кода добавляется новая функция.

Во многих случаях оригинальный исходный код переформатируется; мы добавили разрывы строк и переработали отступы, чтобы обеспечить доступное пространство страницы в книге. В редких случаях даже этого было недостаточно, и списки содержат маркеры продолжения строки (➥). Кроме того, комментарии в исходном коде часто удаляются из листингов, когда описание кода есть в тексте. Аннотации к коду сопровождают многие листинги, выделяя важные понятия.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com или www.дмк.рф на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Manning очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Об авторе

Бен Фаррелл – опытный старший разработчик в компании Adobe, работающий в команде разработчиков прототипов Adobe Design. Бен вместе со своей командой помогает формировать и реализовывать пользовательский опыт продуктов и функций на промежуточном уровне между проектированием и разработкой. Всю свою карьеру он занимался веб-технологиями, но также работал над отмеченными наградами проектами с использованием самых разных платформ и языков.

Об иллюстрации на обложке

Рисунок на обложке книги «Веб-компоненты в действии» озаглавлен как *Bourgeois de Londre* (лондонский буржуа). Эта иллюстрация взята из коллекции костюмов разных стран Жака Грассе де Сен-Совёра (1757–1810) под названием *Costumes Civils Actuels de Tous le Peuples Connus*, опубликованной во Франции в 1788 году. Каждая иллюстрация прекрасно нарисована и раскрашена вручную. Богатое разнообразие коллекции Грассе де Сен-Совёра ярко напоминает нам о том, как культурно были отделены города и регионы мира всего 200 лет назад. Изолированные друг от друга люди говорили на разных диалектах и языках. На улицах или в сельской местности было легко определить, где они жили и какова была их профессия или положение в жизни, по их одежде.

С тех пор наш стиль одежды изменился, и разнообразие по регионам, столь богатое в то время, исчезло. Сейчас трудно отличить жителей разных континентов, не говоря уже о разных городах, регионах или странах. Возможно, мы обменяли культурное разнообразие на более разнообразную личную – определенно, на более разнообразную и динамичную технологическую жизнь.

В то время когда трудно отличить одну компьютерную книгу от другой, издательство Manning празднует изобретательность и инициативу компьютерного бизнеса с помощью обложек книг, основанных на богатом разнообразии региональной жизни двух веков назад, возвращенной к жизни рисунками Грассе де Сен-Совёра.

Часть I

Первые шаги

В последнее время вы, наверное, все чаще и чаще слышали о веб-компонентах. Во многом это связано со всеми популярными, современными браузерами, которые стали поддерживать их в последние месяцы. Это относится и к Microsoft Edge, поскольку вы уже можете скачать предварительный просмотр для разработчиков, пока мы ожидаем официального релиза с поддержкой Chromium. Впрочем, это может немного сбить с толку, если вы посмотрите глубже, чтобы увидеть, что такое веб-компоненты!

Мало того что набор стандартов, составляющих веб-компоненты, немного изменился с течением времени, в действительности веб-компонент можно создать с помощью пользовательских элементов как таковых! Вы можете создать собственный элемент, который будет находиться на вашей HTML-странице, как и любой другой элемент, предоставляемый браузером. Что еще более важно, с помощью API пользовательских элементов ваш элемент может получить специальную логику, чтобы стать полнофункциональным, крошечным интерактивным компонентом, который выглядит простым снаружи и может работать вместе с любым другим элементом на странице.

В первой части этой книги мы расскажем, как создать свои первые пользовательские элементы, а также о некоторых передовых практиках, касающихся них. В конце первой части, даже просто исследуя эту единственную концепцию, вы создадите веб-компоненты, которые действительно полезны в реальных ситуациях, позволяя им быть обернутыми как единый фрагмент, управляя своими собственными зависимостями, возможно, включая другие вложенные веб-компоненты, готовые для размещения на HTML-странице.

1

Фреймворк без фреймворка

Эта глава охватывает следующие темы:

- что такое веб-компонент;
- теневая модель DOM;
- пользовательские элементы;
- библиотеки Polymer и X-Tag;
- функции ES6/ES2015.

Привет, и спасибо за то, что читаете эту книгу! Вот уже несколько лет я использую веб-компоненты практически во всех проектах веб-разработки, которые у меня были.

Будучи веб-разработчиками, наша работа заключается в выборе правильных инструментов для любого конкретного проекта. Это может стать сложной задачей, потому что важны не только насущные потребности проекта. Потребности вашей команды также важны, а еще важен вопрос о том, является ли проект частью более обширной экосистемы в вашей компании, как он будет поддерживаться и как *долго* его нужно поддерживать. Список можно продолжить.

Конечно, эти решения не являются уникальными для веб-разработчиков, но одним из основных отличий между нами и многими разра-

ботчиками программного обеспечения является то, что веб-сообщество выпустило изумительное количество инструментов, библиотек и фреймворков. Может быть, трудно не отставать от них всех – настолько, что уже давно обсуждается «усталость от фреймворков».

Принятие этих новых инструментов, кажется, происходит с молниеносной скоростью. Отложив на мгновение фреймворки, даже что-то такое же нишевое, как и исполнители задач для создания JavaScript-проектов, за последние несколько лет кардинально изменилось. Я был свидетелем перехода от Grunt в 2012 году к Gulp всего пару лет спустя, и теперь наблюдается тенденция действовать по минимуму при использовании Node.js NPM (Node Package Manager) для запуска сценариев сборки. Говоря о менеджерах пакетов, мы, разработчики, колебались, делая выбор между NPM, Bower и Yarn, для запуска наших зависимостей пользовательского интерфейса.

Инструменты сборки и менеджеры пакетов – это одно. Это небольшие, но важные части нашего процесса веб-разработки. Тем не менее такая же ситуация происходит с тем, как мы на самом деле создаем свои приложения и пользовательский интерфейс, что, возможно, является самой главной и важной частью веб-разработки.

Определенным разработчикам может быть трудно поспеть за этим, хотя изучать новый фреймворк или библиотеку интересно. Некоторые из них имеют более крутую кривую обучения, чем другие, и во многих случаях вы изучаете «систему» фреймворка, а не фундаментальные концепции HTML, JS и CSS.

Если речь идет о разработчике в команде или компании, тут есть дополнительные проблемы. В начале проекта вам нужно будет договориться о том, какие инструменты вы будете использовать для разработки в течение жизненного цикла проекта. Это включает в себя инструменты сборки, тестирования и, конечно же, любые фреймворки или библиотеки. Не каждый согласится с лучшим вариантом. Если команда большая и работает над большим количеством проектов, может быть заманчиво позволить разработчикам выбирать свои собственные инструменты для каждого проекта. В конце концов, полезно проанализировать потребности проекта и использовать соответствующие инструменты. Но таким образом мы также игнорируем неизбежное, когда разработчики должны работать сообща, чтобы создать общие части пользовательского интерфейса или интегрировать недавно принятую дизайн-систему, обязательную для всей компании. В конце концов, использование различных инструментов и фреймворков может выйти вашей команде боком.

Если все соглашаются, охотно или нет, использовать один и тот же фреймворк, какое-то время все может быть замечательно. Даже тогда, спустя два или три года, фреймворки могут устаревать. Использование старых технологий начинает сказываться, особенно для начинающих разработчиков в вашей команде, которые хотят, чтобы их навыки соответствовали всему остальному веб-сообществу. В этот момент ваша организация сталкивается с выбором: переделать весь комплекс тех-

нологий, используя новый фреймворк, или оставить старый и столкнуться с ощущением, что это не является инновационным местом для работы.

Это несомненно сложная проблема и решение! Конечно, возникает вопрос: «Какова альтернатива?» Я говорил с довольно многими людьми, которые хотят освободиться от этой постоянной мешанины с фреймворками по ряду причин. «Почему нельзя просто использовать простой HTML, JS и CSS?» Это распространенный вопрос. Одним из главных преимуществ отказа от фреймворка является возможность сосредоточиться на основных концепциях веб-разработки, а не на изучении специфических для фреймворка навыков, которые можно или нельзя перенести в следующий популярный фреймворк. Еще одним огромным преимуществом является возможность опробовать небольшие библиотеки и микрофреймворки, которые решают конкретные задачи в вашем проекте. Барьер входа в эти и даже новые инструменты сборки пользовательского интерфейса намного ниже, учитывая, что вы не боретесь с конкретной средой разработки, предоставляемой последним популярным фреймворком.

Современные фреймворки чрезвычайно полезны и решают большие проблемы, но почему мы больше не слышим об использовании так называемого «чистого JavaScript», учитывая желание разработчиков попробовать что-то другое? В некоторой степени слышим. Рассмотрим результаты опроса, проведенного State of JavaScript в 2017 году: https://2017.stateofjs.com/2017/front-end/results/. Вы увидите, что разработка без применения фреймворков занимает второе место по популярности, уступая только React.

Тем не менее мы не знаем, почему люди утверждают, что предпочитают не использовать фреймворк, а чистый JS. Что создают эти разработчики? Какие инструменты/процессы они используют? Мне было бы любопытно узнать, создают ли они что-то вроде фреймворка для решения проблемы отсутствия структуры и организации кода, что обычно предоставляют современные фреймворки.

Этот последний момент, касающийся структуры и организации кода, заключается в том, что веб-разработка без фреймворка была для меня незапланированной в прошлом, и именно поэтому я всегда обращался к последним фреймворкам. Без структуры ваш код превращается в спагетти. Сохранение и написание новых функций может стать безумием без предсказуемой организации проекта. Тем не менее мне хотелось освободиться от больших, полномасштабных фреймворков; когда я впервые увидел веб-компоненты, я увидел огромную возможность, позволяющую сделать именно это.

Итак… как это сделать? Чтобы действительно решить этот вопрос, нужно понять, что такое веб-компоненты. Прежде чем углубиться в детали, мы будем использовать календарь с возможностью выбора даты в качестве примера, с которым мы все, вероятно, сталкивались. Хотя это и не веб-компонент, по сути, это аналогичная концепция, если заглянуть внутрь.

1.1 Что такое веб-компоненты?

Современные популярные фреймворки сегодня в основном предлагают возможность многократного использования кода в виде *компонентов*, или *модулей*. В целом это совместно используемые и автономные фрагменты кода (HTML/JS/CSS), которые предлагают визуальный стиль и интерактивность и, возможно, имеют API или параметры, которые можно настраивать.

Подумайте о том, что уже есть в вашем браузере. И учтите, что у нас уже есть многократно используемые модульные элементы, которые предлагают стиль и интерактивность и поставляются с API.

Конечно, я говорю о тегах HTML, или элементах DOM. Они отображаются в DOM и имеют определенный тип функциональности. Теги `<div>` или `<span>` достаточно универсальны и используются для хранения текста или смеси элементов. Элементы `<button>` или `<input>` более специфичны по функциональности и стилю. Когда вы помещаете кнопку в свой HTML-код, она выглядит как стандартная кнопка, а когда вы нажимаете на нее, она ведет себя как кнопка. Это похоже на различные стили `<input>`, независимо от того, хотите ли вы создать календарь с возможностью выбора даты, ползунок или поле ввода текста.

1.1.1 Календарь с возможностью выбора даты

Возьмем, к примеру, календарь с возможностью выбора даты. Чтобы создать его, просто поместите приведенный ниже тег в HTML-код:

```
<input type="date">
```

Выглядит легко, не так ли? Так и есть! То, что вы на самом деле получаете из этого простого тега, довольно сложно, но все это обрабатывается вашим браузером. Этот тег (при использовании типа `"date"`) предлагает поле для ввода текста, и вы можете щелкнуть по месяцу, дню или году и перейти вверх или вниз. Кроме того, если щелкнуть по стрелке, указывающей вниз, откроется представление календаря, с которым пользователь может взаимодействовать, чтобы выбрать дату, как показано на рис. 1.1. Следует отметить, что на мобильном телефоне он действует немного иначе. Он не будет открываться, как это происходит в настольном браузере. Вместо этого появится модальное окно.

Более того, этот календарь имеет *свойства*, которые можно запросить, включив значение. В этом можно убедиться, зарегистрировав свойство в консоли JS:

```
console.log( document.querySelector('input').value );
```

Когда я регистрирую его, то вижу текущее значение календаря в своей консоли. Он также отправляет *события*, которые я могу прослушать, когда значение изменяется или отправляется. Я также могу вызывать *методы* для календаря для перехода по датам.

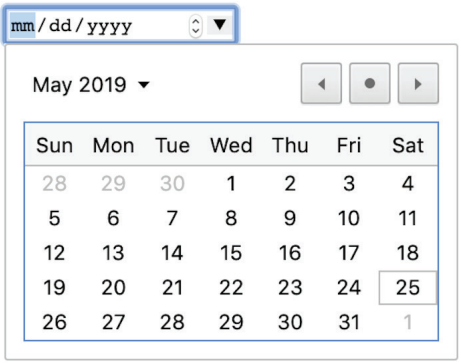

Рис. 1.1 Расширенный пользовательский интерфейс календаря с возможностью выбора даты

Календарь с возможностью выбора даты является отличным примером повторно используемых компонентов или модулей с довольно сложным визуальным стилем и шаблонами взаимодействия, которые должны программироваться компаниями-разработчиками браузеров. Они работают в разных ситуациях. Также этот календарь представляет собой отличный пример популярной концепции веб-компонентов под названием Shadow DOM (Теневая модель DOM).

1.1.2 Теневая модель DOM

Теневая модель DOM – это способ изолировать ваш веб-компонент и предотвратить непреднамеренные последствия более крупного приложения. Когда вы откроете инструменты разработчика для просмотра DOM, то просто увидите тег `<input type="date">`. Однако если вы используете браузер Chrome и активируете пункт «Показывать теневую модель DOM агента пользователя» в настройках инструментов разработчика, тот же самый тег `<input>` приобретает иной вид и выглядит, как показано на рис. 1.2.

Под `#shadow-root` намного больше разметки! Лично первое, на что я обращаю внимание при изучении этого кода, – всплывающее окно календаря. Хотя было бы замечательно увидеть этот фрагмент в HTML и CSS, его там нет, потому что этот фрагмент пользовательского интерфейса является частью вашей собственной ОС, которую ваш браузер просто показывает посредством элемента.

Тем не менее у нас есть значительное количество элементов, скрытых в нашей «теневой» DOM, которые все появляются в элементе поля ввода.

Если присмотреться, можно заметить, что в нашей теневой модели DOM размещены теги `<div>` и `<span>`. Вам может прийти в голову, что это опасно! Почему? Что же, в таблице стилей CSS своего приложения я вполне мог бы сделать так, чтобы во всех тегах `<div>` был указан синий фон с очень большим размером шрифта, а все теги `<span>` отображались с прозрачностью 10 %. Если бы вы не знали о существовании этой дополнительной разметки, вы могли бы случайно испортить все календари с возможностью выбора даты, за исключением одной важной вещи: теневая DOM защищает внутреннюю работу вашего веб-компонента из-

вне. Стили ваших синих/больших тегов `<div>` не будут проникать в теневую DOM. Более того, вы не сможете написать код Java Script, чтобы попытаться заполучить кнопку календаря `clear` и манипулировать ей:

```
let myElement = document.getElementById('clear');
```

```
▼<input type="date" name="bday">
  ▼#shadow-root (user-agent)
    ▼<div pseudo="-webkit-datetime-edit" id="date-time-edit" datetimeformat=
    "M/d/yy">
      ▼<div pseudo="-webkit-datetime-edit-fields-wrapper">
          <span role="spinbutton" aria-placeholder="mm" aria-valuemin="1"
          aria-valuemax="12" aria-label="Month" pseudo="-webkit-datetime-edit-
          month-field">mm</span>
          <div pseudo="-webkit-datetime-edit-text">/</div>
          <span role="spinbutton" aria-placeholder="dd" aria-valuemin="1"
          aria-valuemax="31" aria-label="Day" pseudo="-webkit-datetime-edit-
          day-field">dd</span>
          <div pseudo="-webkit-datetime-edit-text">/</div>
          <span role="spinbutton" aria-placeholder="yyyy" aria-valuemin="1"
          aria-valuemax="275760" aria-label="Year" pseudo="-webkit-datetime-
          edit-year-field">yyyy</span>
        </div>
      </div>
      <div pseudo="-webkit-clear-button" id="clear" style="opacity: 0;
      pointer-events: none;"></div>
      <div pseudo="-webkit-inner-spin-button" id="spin"></div>
      <div pseudo="-webkit-calendar-picker-indicator" id="picker"></div>
</input>
```

Рис. 1.2 Активирование соответствующих настроек в инструментах разработчика Chrome позволяет нам увидеть скрытую теневую модель DOM тега input

Когда мы пытаемся получить этот элемент, потому что он находится в пределах теневой DOM, оказывается, что элемент не найден, и наша переменная `myElement` имеет значение null. На рис. 1.3 показаны различные попытки с CSS и JS.

Таким образом, Shadow DOM защищает область видимости корня теневого дерева (shadow root). Да, вы можете использовать его, где угодно. Но это имеет огромное значение в пользовательском элементе, который вы создали, чтобы избежать непреднамеренной поломки, когда разработчик устанавливает правило CSS, имя которого совпадает с именем, которое вы использовали в своем компоненте, или когда тот же разработчик запрашивает элемент по классу и что-то в вашем пользовательском элементе выбирается случайно.

Как вы можете себе представить, календарь с возможностью выбора даты является полезным элементом для дополнения нескольких других полезных элементов, которые мы применяем ежедневно. Многие элементы используются в семантических целях, например тег `<footer>`, а другие имеют определенный API и стиль, например теги `<button>`, `<option>` и `<video>`.

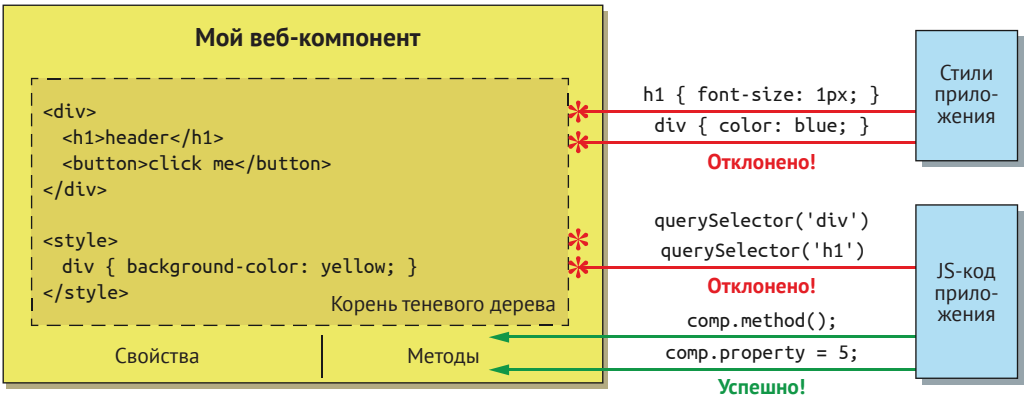

Рис. 1.3 Теневая DOM защищает ваш компонент от непреднамеренных последствий, когда CSS или JS может повлиять на стили и узлы внутри, не подлежащие изменению. Вместо этого у вашего компонента будет пользовательский API, с которым можно будет взаимодействовать, применяя методы и свойства

1.1.3 Что имеют в виду, когда говорят «веб-компоненты»?

Каким бы чудесным ни был календарь с возможностью выбора даты и любой другой элемент, разве не было бы удивительно, если бы мы могли создавать свои собственные элементы с собственным визуальным стилем, внутренней логикой, возможностью повторного использования и инкапсуляцией?

Это то, что люди имеют в виду, когда говорят о веб-компонентах. В дополнение к инкапсуляции, предоставляемой Shadow DOM, мы можем использовать API пользовательских элементов для создания собственных компонентов, которые выполняют функции, соответствующие нашим потребностям.

Для меня это обещание веб-компонентов. Я хочу взять что-то, что меня интересует, и создать многократно используемый фрагмент, которым могу поделиться со всем миром, своей командой или просто с самим собой, чтобы использовать его в нескольких проектах там, где мне это нужно. С другой стороны, может существовать фрагмент интерфейса, который я нахожу скучным создавать снова и снова. С помощью веб-компонентов я могу создать его один раз, использовать в нескольких проектах и дополнять по мере необходимости. И даже лучше: возможно, кто-то еще создал веб-компонент для того, что нужно мне, и у меня нет времени или опыта, чтобы воссоздать его. Он может поделиться им со мной, и я могу просто использовать его как обычный элемент DOM.

1.1.4 Проблемная история импорта HTML

К сожалению, некоторые в сообществе веб-разработчиков считают обещание веб-компонентов нарушенным. Я, конечно, не могу винить их за это. Говоря о конкретных технических функциях, предлагаемых веб-

компонентами, видение начало разваливаться после того, как первоначальная шумиха вокруг веб-компонентов поутихла несколько лет назад.

Приблизительно в 2015 году было широко известно, что стандартный веб-компонент будет создаваться с использованием трех новых функций:

- пользовательские элементы;
- Shadow DOM;
- импорт HTML.

Пока я еще даже не упомянул об импорте HTML. Эта концепция так и не была принята в качестве стандарта. Фактически вначале Google была в значительной степени ответственна за создание рабочих проектов веб-компонентов. Google взяла на себя задачу создавать API-интерфейсы и отправлять их в Chrome в качестве обнадеживающего эксперимента, чтобы посмотреть, будут ли запущены веб-компоненты. Однако ничего не получилось; другие разработчики браузеров в то время не планировали поставлять эту функцию. Firefox, в частности, хотел подождать, чтобы увидеть, насколько большую сенсацию произведут модули ES6/ES2015, и, возможно, когда-нибудь импортировать не только JS, но и HTML.

Импорт HTML был довольно большой потерей. С самого начала планы Google по доставке веб-компонентов зависели от него. Импорт HTML, как показано на рис. 1.4, был фрагментом HTML-кода для объявления разметки или структуры компонента, а также включал в себя код JS, определяющий логику компонента. Импорт HTML был основной точкой входа для веб-компонентов, и без них мы были в растерянности относительно того, как использовать веб-компоненты с разметкой и стилем вообще.

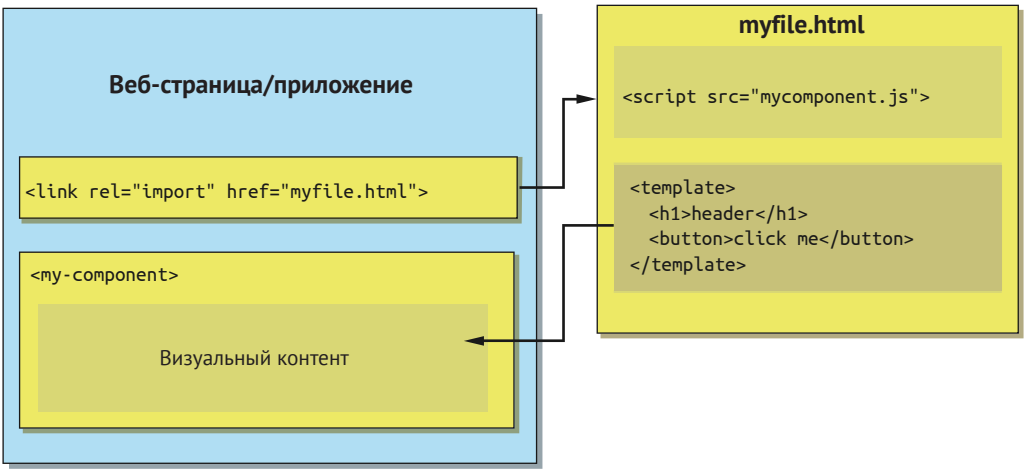

Рис. 1.4 С помощью импорта HTML-файл, содержащий определение вашего компонента и его разметку, можно импортировать прямо в ваш документ

Shadow DOM был не намного лучше в то время. Chrome был единственным браузером, принявшим его. Firefox принял его только в октяб-

ре 2018 года, и мы ожидаем, что Microsoft Edge выпустит его, хотя уже доступна предварительная версия для разработчиков.

Shadow DOM и API пользовательских элементов также перешли с версии 0 на 1. Для пользовательских элементов это было несколько проблемным, учитывая, что разработчикам, которые были знакомы с веб-компонентами в это непростое время, было предложено перейти на новый API.

Учитывая все это, вряд ли можно критиковать разработчиков, которые называли веб-компоненты «нарушенным обещанием» и переключались на фреймворк. Могу заверить, что в 2015 году было немного сложно правильно с ними работать, особенно при работе с браузерами, отличными от Chrome.

1.1.5 Библиотеки Polymer и X-Tag

Еще одним аспектом того, что имеется в виду, когда речь идет о веб-компонентах, были библиотеки, появившиеся в то время, когда использовали веб-компоненты в качестве основы. Из-за нестабильности, связанной с простыми компонентами без использования фреймворков в то время, библиотеки Polymer от компании Google (https://polymer-library.polymer-project.org) и X-Tag от Mozilla (https://x-tag.github.io) были тем, что люди считали веб-компонентами, или, по крайней мере, единственным способом работы с ними.

Библиотека Polymer проделала большую работу по продвижению стандартов и рабочих процессов, и теперь похоже, что 3.0 является последней официальной версией функции, поскольку она переходит в режим обслуживания. Вместо этого команда разработчиков разбивает некоторые основные инструменты на гораздо более мелкие и более целевые решения, такие как lit-html и LitElement в рамках проекта Polymer. Эти основные инструменты и функции хорошо согласуются с подходом, не предусматривающим использование фреймворков, который я описал в этой книге.

Несмотря на то что команда проделала отличную работу над серией надежных выпусков и сейчас работает над тем, чтобы сосредоточиться на более мелких и дополнительных функциях, в первые дни библиотека Polymer до выхода версии 1.0 была слегка шаткой. Как и ожидалось с любой библиотекой до версии 1.0, API-интерфейсы немного изменились, тем более что они пытались не отставать от меняющихся спецификаций и отсутствия Shadow DOM в каждом браузере, кроме Chrome. С Shadow DOM было особенно трудно иметь дело. Полнофункциональные полифилы, включающие инкапсуляцию CSS, были слишком трудными и влияли на производительность. Чтобы компенсировать это, в качестве облегченной реализации был придуман Shady DOM, который можно было использовать для полизаполнения.

Это было непростое время для веб-компонентов в целом, и библиотека Polymer казалась еще одним фреймворком или библиотекой, которая должна была конкурировать с более надежными библиотеками, не имевшими отношения к промежуточным веб-стандартам.

1.1.6 Современные веб-компоненты

Несмотря на эти тяжелые времена, я остановился на веб-компонентах. Я успешно использовал их для проектов, но не был полностью удовлетворен, пока не начал использовать некоторые новые функции языка JS. Функция жирной стрелки оказалась отличным способом управления областью видимости при работе с событиями мыши или таймерами. Что еще более важно, ключевое слово `import` и концепция модулей были огромными.

С помощью этого ключевого слова я смог отойти от хрупкого беспорядка, когда мне нужно было убедиться, что каждый JS-файл, который я хотел использовать, был привязан к тегу `<script>` на моей главной HTML-странице. Каждый веб-компонент может нести полную ответственность за импорт собственного кода. Это означало, что на главной HTML-странице я мог заставить один тег `<script>` на основе модуля импортировать веб-компонент, который содержал все мое приложение. Каждый дочерний компонент просто импортирует все, что ему нужно.

Это открыло возможности для многократно используемых модулей кода, написанных на чистом JS, и дало мне возможность создавать несколько уровней наследования, когда я хотел, чтобы мои компоненты имели общий API и были немного умнее базового API `HTMLElement`. Наконец, я мог хранить свой HTML/CSS в отдельном файле template.js, который мог импортировать, отделяя свои визуальные проблемы от логики контроллера компонента.

Последней огромной функцией JS, которая сделала работу с веб-компонентами приятным удовольствием, был *шаблонный литерал*. Я не только мог хранить свой HTML/CSS в отдельном файле шаблона, но и мог заменить выражения-заполнители в своей разметке переменными и вкладывать несколько шаблонов вместе, используя функции Java Script.

Эти функции ES6/ES2015 неожиданно сделали веб-компоненты приятными для работы.

Даже ранее, работая с ныне устаревшим HTML-импортом, я считаю, что сочетание модулей и шаблонных литералов – гораздо лучший путь, если сравнивать.

Как я уже говорил, Shadow DOM поддерживается на 99 %. На это ушло какое-то время, но все разработчики популярных браузеров поддерживают эту технологию. Мы просто ждем, когда Microsoft выпустит предварительную версию Edge для всех. Лично я только что начал работать с Shadow DOM, после того как ее стал поддерживать Firefox.

В то же время какой бы хорошей ни была технология Shadow DOM, она также не является обязательной. Правда, она дает дочерним элементам нашего компонента хорошую защиту от проникновения стилей и JS, что приводит к неблагоприятным последствиям, но это новое решение проблемы, которая всегда у нас была. Поэтому, если нам нужно подождать поддержки браузера несколько месяцев или просто отказаться от него в краткосрочной перспективе, это еще не конец света. Тем не менее я достаточно долго сдерживал свое волнение относительно Shadow DOM

из-за предыдущей поддержки в браузере; теперь, когда мы собираемся пересечь финишную черту, я взволнован, потому что оказывается, что это так здорово – использовать данную технологию.

Как бы я ни волновался за будущее веб-компонентов, я не слышал ни о каком современном видении относительно них, особенно что касается разработчиков, которых ранее они приводили в смятение. Если бы мне пришлось переопределить «обещание веб-компонентов» на 2019 год, это бы уже не были те три обязательных свойства: пользовательские элементы, Shadow DOM и импорт HTML.

Для меня видение веб-компонентов 2019 года складывается из набора инструментов функций ES6/ES2015 и тега `<template>`, когда, и если вам это нужно, все они служат пользовательскому элементу в качестве основной функции. Как только технология Shadow DOM будет поддерживаться во всех браузерах в ближайшем будущем, она также станет важным дополнением к нашему набору инструментов. Это видение того, как я буду подходить к веб-компонентам в данной книге. Мы подробно рассмотрим пользовательский элемент, а затем изучим рабочие процессы вокруг всех дополнительных инструментов в нашем наборе.

1.2 Будущее веб-компонентов

Предсказывать будущее всегда непросто, особенно это касается интернета, где все меняется в безумном темпе. Тем не менее у нас есть несколько убедительных подсказок, указывающих на то, что может произойти с веб-компонентами после 2019 года.

Мы уже видели эксперименты с React, Angular (https://angular.io/guide/elements) и Vue (https://vuejsdevelopers.com/2018/05/21/vue-js-web-component/), где показана компиляция компонентов в каждом из этих фреймворков в отдельный веб-компонент, работающий абсолютно независимо от фреймворка, который сделал эти компоненты. Кроме того, такие инструменты, как StencilJS (https://stenciljs.com) и Svelte (https://svelte.technology), позволяют использовать фреймворк и выполнять компиляцию в автономные веб-компоненты.

Что это значит? Вскоре мы все можем создавать компоненты без какого-либо фреймворка или с помощью фреймворка на свой выбор. Мы будем использовать веб-компонент, созданный React в Angular, или веб-компонент, созданный Vue, на веб-странице без фреймворка. Искусственные стены между разработчиками и их фреймворками могут относительно скоро разрушиться, как показано на рис. 1.5. И все это благодаря веб-компонентам.

Эта концепция может даже распространяться на совместную работу совершенно разных языков. Одно приложение может иметь разные компоненты, разработанные в JS, Typescript и CoffeeScript; учитывая, что каждый из них является модульным компонентом, предоставляющим API, это не имеет значения. Что еще более безумно, с появлением WebAssembly мы могли видеть, что в таких языках, как C ++, Lua, Go и т. д.,

код, скомпилированный в байт-код и обернутый веб-компонентом, выглядит как совершенно нормальный элемент извне, одновременно позволяя получать высокопроизводительную графику, которая может работать быстрее, чем обычно в JS.

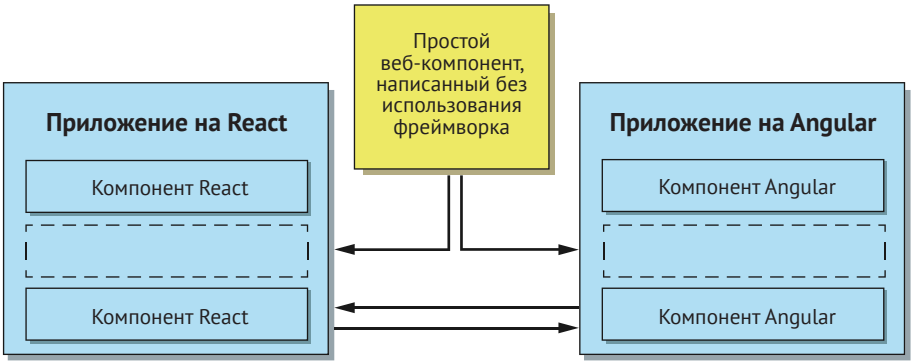

Рис. 1.5 Веб-компоненты могут в будущем преодолеть разрыв между популярными фреймворками. В этих фреймворках можно использовать не только веб-компоненты без фреймворка, но уже существующие экспериментальные проекты по компиляции компонентов в React, Angular или Vue для независимого запуска компонентов, которые можно применять где угодно

Я также думаю, что использование модулей ES6/ES2015 и импорта изменит наш взгляд на библиотеки и фреймворки. Мы уже видим два похожих инструмента, lit-html и hyperHTML, для расширенного управления разметкой. Оба из них имеют модули, которые разработчики могут импортировать и вместо того загружать целую библиотеку для решения целевой проблемы. Вам разрешается участвовать или отказаться от участия в любое время, когда захотите, на протяжении своего проекта.

В этой связи я думаю, мы увидим гораздо больше удивительных библиотек. Вы будете импортировать только то, что вам нужно и когда вам это нужно. Людям может быть скучно с веб-компонентами как новой блестящей парадигмой, но я вижу, как мы опираемся на эти основы с помощью импортируемых скриптов и библиотек. Новый подход проекта Polymer, когда команда переводит свою исходную библиотеку в режим обслуживания, кажется, точно соответствует этому. Время покажет, будут ли основные фреймворки разделять функции, как это сделала команда Polymer в случае с lit-html, на отдельные операции импорта, которые можно использовать вне фреймворка. Но мне это кажется неизбежным, особенно если смотреть на другие языки, которые всегда обладали функционалом импорта.

1.3 За пределами одного компонента

До сих пор я много говорил о веб-компонентах как об отдельных компонентах, но как бы я ни любил автономные веб-компоненты, они не будут

особенно полезны, если не будут работать вместе для создания вашего приложения.

Задолго до того, как появились веб-компоненты, у нас были отличные способы взаимодействия с обычными элементами DOM. Мы можем использовать те же методы, чтобы придать структуру всему, что мы создаем с помощью веб-компонентов, так же как делаем это с обычными тегами `<div>`, `<video>` или `<input>`.

1.3.1 Веб-компоненты как и любой другой элемент DOM

Для начала каждый элемент имеет своего рода публичный API. Под этим я подразумеваю, что вы можете получать и устанавливать свойства для своего элемента и вызывать функции. Например, в случае с элементом video вы можете вызывать функции `pause()` и `play()` для управления воспроизведением видео. Вы также можете уточнить продолжительность видео, проверив свойство `duration`. Наконец, для того чтобы перескочить на определенный момент в своем видео, можно настроить свойство `currentTime`.

Очевидно, что методы и функции для объектов распространены повсеместно в программировании. Элементы DOM ничем не отличаются, как можно увидеть по рис. 1.6; более того, пользовательские веб-компоненты также не являются исключением.

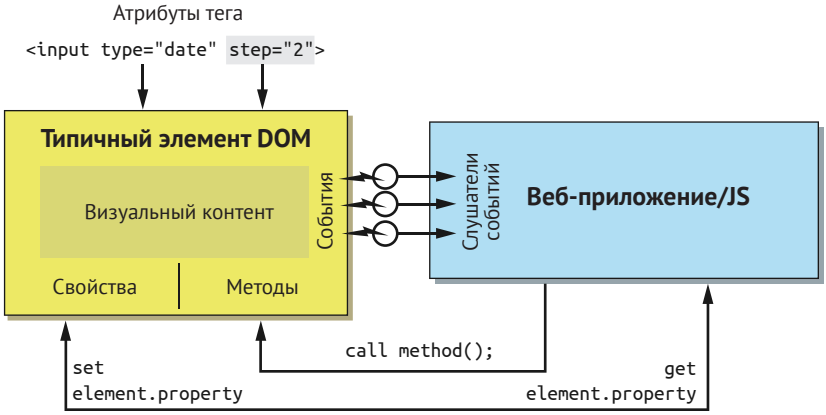

Рис. 1.6 DOM-элементы имеют различные свойства, методы, события и атрибуты, которые используются, чтобы сообщать элементу, как действовать и взаимодействовать с внешним миром

Несколько похожими на свойства являются *атрибуты*. Вы видите их все время в HTML-коде. У такого простого тега, как `<img>`, есть атрибут `src`, который указывает на местоположение изображения. Атрибуты представляют собой простую концепцию, но они удобны для предоставления вашему веб-компоненту различного поведения в зависимости от того, как вы хотите, чтобы он работал. Более того, у веб-компонентов есть API-интерфейс, позволяющий внутренне прослушивать изменения в атрибутах.

В предыдущем примере с элементом video атрибуты, предоставляемые тегом, не соответствуют свойствам, предоставляемым API. Хотя мы можем настроить свойство `currentTime`, мы не можем установить тот же атрибут для тега. В противовес этому неоднократно в случае с веб-компонентами, которые вы создаете, у вас будет возникать желание использовать передовой метод *отражения*. При настройке свойств вам понадобится обновить атрибут (и наоборот), чтобы эти атрибуты и свойства были синхронизированы. Конечно, это не жесткое правило, а всего лишь общепринятый передовой метод. До появления веб-компонентов не обязательно было придерживаться отражения. Хорошим примером того, когда что-то может пойти не так, является атрибут `value` тега `<input>`. Здесь этот атрибут устанавливает начальное значение, но при его изменении атрибут остается прежним. Запрос свойства `value` посредством JS вернет самое последнее значение, при условии что оно было изменено. Это сбивает с толку! Но мы просто принимаем это, потому что так всегда работал тег `<input>`. При создании новых веб-компонентов, вероятно, лучше избегать этой путаницы и отражать атрибуты и свойства. В этом смысле атрибут или свойство `muted` элемента `video` является хорошим примером отражения.

Наконец, возможно, вам понадобится прослушать изменения из вашего пользовательского веб-компонента. Мы постоянно используем события в других сценариях. Возьмем, к примеру, нажатие на кнопку. Как правило, для прослушивания клика мы делаем следующее:

```
mybutton.addEventListener('click', functionToCall );
```

Вы также можете создавать и отправлять собственные пользовательские события. Это можно делать из любого места, но они особенно удобны, когда вам нужно, чтобы ваше приложение или другие компоненты внутри него слушали события, поступающие от вашего веб-компонента.

1.3.2 От отдельного компонента к приложению

Говорить об отдельных компонентах – это одно, но что делать, когда вам нужно создать целое веб-приложение? Веб-компоненты могут быть настолько большими или маленькими, как вам нужно. Вы можете создать несколько чрезвычайно детализированных компонентов, таких как кнопки, а затем вложить их в более крупный веб-компонент, например в пользовательскую панель инструментов.

Ваш компонент панели инструментов может обрабатывать более мелкие детали работы с кнопками, возможно, включать и выключать их или отключать определенные кнопки при определенных обстоятельствах.

Нашу панель инструментов наряду с другими компонентами, показанными на рис. 1.7, можно дополнительно вложить в другой родительский компонент и т. д. Это может продолжаться до тех пор, пока в вашем теге `<body>` не будет единственного веб-компонента.

Веб-компоненты и JavaScript без фреймворка могут многое предложить для разработки веб-приложений. Но по мере роста вашего прило-

жения растет и его сложность. Координировать взаимодействие ваших компонентов друг с другом становится все труднее.

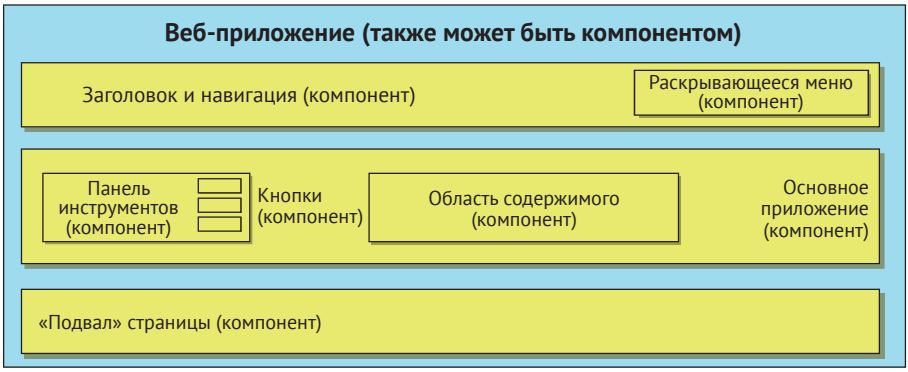

Рис. 1.7 Пример веб-приложения, состоящего из веб-компонентов, которые сами состоят из большего количества веб-компонентов. Эта иерархия может распространяться на нечто маленькое, как пользовательская кнопка, или быть таким же большим, как все приложение, обернутое как веб-компонент

Иногда можно обнаружить, что даже с присущей вам структурой, которую дают веб-компоненты, этого недостаточно для создания вашего комплексного приложения. Возможно, вы захотите обратиться к популярным фреймворкам и библиотекам, чтобы они помогли вам со структурированием. Такие фреймворки, как Angular, предлагают привязку данных, шаблоны MVC и многое другое. Конечно, они могут быть полезны при создании традиционного веб-приложения. С другой стороны, мы можем написать и импортировать простой код JS на базе проверенных временем шаблонов проектирования, которые уже давно используются, избегая этих более крупных фреймворков.

Например, собственные события DOM могут не сработать в вашем случае. Зачастую вам нужно, чтобы одна часть вашего веб-приложения передавала сообщение совершенно другой части вашего приложения, и вам не нужно беспокоиться о том, как происходит событие в DOM. Вы *могли бы* обратиться к такой библиотеке, как RXjs или Redux, но это может быть излишним. Вместо этого можно написать простую шину событий с небольшим количеством кода. На рис. 1.8 и 1.9 сравниваются два этих подхода.

На рис. 1.8 у вас могут, например, быть компоненты ввода формы, содержащиеся в веб-компоненте. Эти компоненты ввода могут инициировать изменения ввода текста, выпадающие изменения и многое другое, все для этого родительского компонента. Хорошим примером этого может быть компонент палитры цветов с вводом цвета RGB и ползунками. Родительский веб-компонент (палитра цветов), в котором размещены эти входные компоненты, должен будет передать цвет своему родительскому веб-компоненту в другом событии, чтобы сообщить шестнадцатеричное значение цвета.

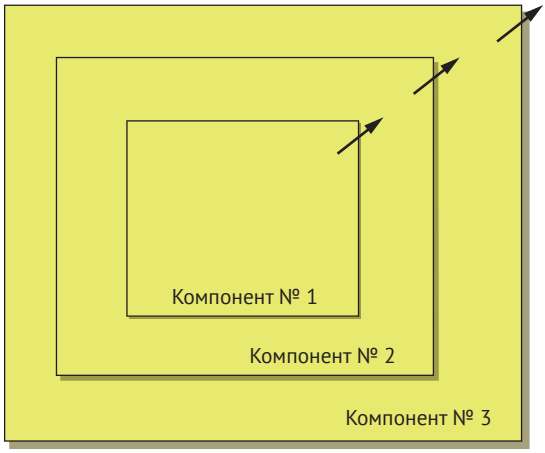

Рис. 1.8 События естественным образом всплывают из вложенных элементов

Это естественное всплывание событий может прекратиться, если то, чей цвет вы решили изменить, находится на другой стороне вашего DOM в ином разделе дерева DOM. В таком случае вам нужно будет использовать иную стратегию, например шину событий (рис. 1.9).

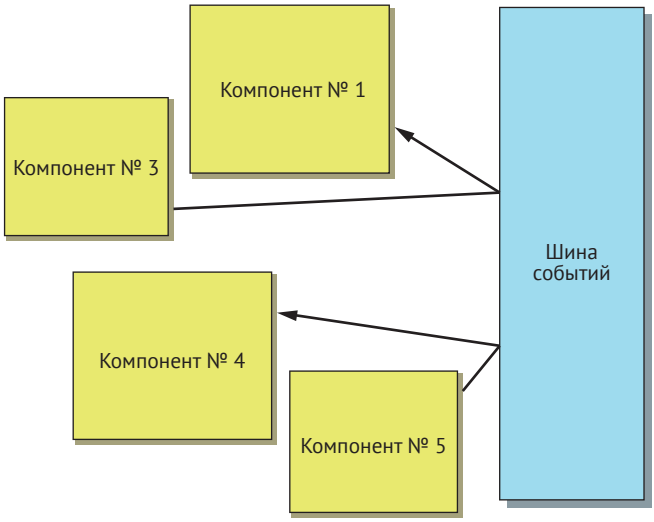

Рис. 1.9 Если обычное всплывание событий нежелательно, используя небольшое количество кода, вы можете создать систему шины событий для маршрутизации событий туда, куда вы хотите

Также можно найти золотую середину, используя микрофреймворки. Микрофреймворки могут быть отличным минималистичным способом организации вашего приложения и добавления определенных функций, не слишком обдумывая его, как это делает более крупный фреймворк. Беспокойство по поводу мельчайших деталей в ваших пользовательских

веб-компонентах, а также организация более крупного приложения с помощью этих небольших библиотек может быть прекрасным способом. Даже минималистичные решения для привязки данных и маршрутизации можно найти и через NPM.

1.4 Ваш проект, ваш выбор

В конце концов, несмотря на то что для использования веб-компонентов без фреймворка есть веские аргументы, ваш проект и ваша команда в конечном итоге будут влиять на то, что вы используете при создании чего-либо для интернета. Как и любой новый стандарт, веб-компоненты пока не дают ответов на все вопросы. Равно как и ни один популярный фреймворк.

Бывают случаи, когда у вас имеется чрезвычайно простое веб-приложение, и современный фреймворк может быть идеальным ответом, поскольку он справляется со всем, что вам нужно сделать. В других случаях вы можете работать над типом проекта, в котором фреймворки просто мешают. Решения, которые вы можете выбрать, охватывают широкий спектр вариантов, причем некоторые из них перекрываются.

Даже если веб-компоненты без фреймворка не являются правильным решением для вас, однажды ваш любимый фреймворк, скорее всего, будет создан с их использованием, хотя это может быть и неочевидно. Знакомство с основами на базе веб-стандартов любого фреймворка – это всегда хорошая идея, даже если вы не используете их напрямую.

Несмотря на несколько запутанный перерыв в работе веб-компонентов несколько лет назад, сейчас мы находимся в том месте, где они являются реальным вариантом для создания вашего следующего проекта.

Я уверен, что в ближайшие годы мы увидим новые идеи и методы для вашего рабочего процесса веб-компонентов, но эти идеи будут основаны на стандартах, которые я буду рассматривать в данной книге, а также на новейших и появляющихся текущих рабочих процессах. Мы будем изучать веб-компоненты на атомарном уровне, вплоть до приложений, созданных из множества компонентов, а также рассмотрим, как управлять вашим HTML и CSS, организовывать проекты, и многое другое. Я надеюсь, что вы так же волнуетесь, как и я, о будущем интернета!

Резюме

Из этой главе вы узнали:

- что за последние несколько лет веб-компоненты превратились из рабочего проекта, принадлежащего Google, в настоящий веб-стандарт, принятый всеми современными браузерами;
- о Shadow DOM как дополнительной, но важной функции, находящейся на грани принятия всеми браузерами;

- о месте веб-компонентов в современных фреймворках, а также независимой части любой экосистемы;
- о потенциальном будущем веб-компонентов с постоянно расширяющимся сообществом модулей JS в духе библиотек Polymer Project, таких как lit-html и litelement, а также других библиотек, таких как hyperHTML;
- об отдельном веб-компоненте в сравнении с целым приложением, состоящим из веб-компонентов.